UNIVERSIDAD AUTÓNOMA DE MADRID

ESCUELA POLITÉCNICA SUPERIOR

Grado en Ingeniería Informática

TRABAJO FIN DE GRADO

DISEÑO E IMPLEMENTACIÓN DE UN CLIENTE TWITTER EXTENDIDO

Eloy Sanchez Moreno
Tutor: Álvaro Ortigosa Juárez

JUNIO 2018

DISEÑO E IMPLEMENTACIÓN DE UN CLIENTE TWITTER EXTENDIDO

AUTOR: Eloy Sanchez Moreno
TUTOR: Álvaro Ortigosa Juárez

Dpto. Ingeniería Informática
Escuela Politécnica Superior
Universidad Autónoma de Madrid
Junio 2018

Resumen (castellano)

El término Fake News es relativamente nuevo y es un producto, normalmente noticias, que se suele difundir en medios de comunicación (prensa escrita, radio, televisión y últimamente en Redes sociales) con el objetivo de engañar o desinformar al usuario que la está leyendo.

En los últimos años, dado el incremento del uso de las Redes Sociales, la aparición de noticias falsas a ido en aumento, ademas las fake news tienen un 70% más de posibilidades de ser retuiteadas que una información real [12], es por eso que herramientas como las que se desarrollan en este proyecto son útiles en este tipo de detección, con el fin de saber en todo momento si lo que se está leyendo puede ser engañoso de algún modo.

Este proyecto consiste en un cliente Twitter extendido donde se mostrarán tweets con un porcentaje de credibilidad.

El sistema está pensado para funcionar en multiplataforma, ya sea en un navegador de un ordenador o mediante una aplicación en IOS o Android.

Conectándose mediante la plataforma Twitter a la cuenta del usuario, se recuperan sus Tweets y se analizan, mediante un proceso externo, para determinar si cada uno de los Tweets contienen noticias falsas. Manteniendo en todo momento actualizada la información de los Tweets y comprobando al instante las características de éste para la detección de fake news. Este proceso de detección externo, ajeno a este proyecto, irá procesando cada tweet y guardando el porcentaje en la base de datos.

El proyecto ha sido implementado mediante la plataforma Meteor, utilizada para la interfaz y tratamiento de datos y mediante MongoDB para el uso de base de datos. También se ha utilizado la API de Twitter para la obtención de los Tweets.

Palabras clave (castellano)

Noticias falsas, Twitter, Tweet, multiplataforma, Redes sociales, Meteor, MongoDB, NodeJS, Javascript, Aplicaciones móviles, bases de datos.

Abstract (English)

The term Fake News is relatively new and is a product, usually news, which is often disseminated in the media (written press, radio, television and lately in social networks) with the aim of deceiving or misinforming the user who is reading it.

In recent years, the increase in the use of social networks, the appearance of false news on the rise, the latest false news have a 70% more chance of being retweeted than real information [12], that is why tools such as those that have been found in this project are useful in this type of detection, in order to know at all times, if what you are doing, can be complicated in some way.

The thesis involves an extended Twitter client where tweets will be shown with a percentage of reliability.

The system is designed to work in multiplatform, either in a computer browser or through an IOS or Android application.

Connecting through the Twitter platform to the user's account, their Tweets are retrieved and analyzed, through an external process, to determine if each of the Tweets contains false news. Keeping at all times the information of the Tweets and checking instantly the characteristics of this for the detection of fake news. This external detection process, unrelated to this project, will process each tweet and save the percentage in the database.

The project has been implemented through the Meteor platform, used for the interface and data processing and through MongoDB for the use of the database. The Twitter API has also been used for the treatment of Tweets.

Keywords (inglés)

Fake news, Twitter, Tweet, multiplatform, Social networks, Meteor, MongoDB, NodeJS, Javascript,

Mobile applications, databases.

Agradecimientos

Quiero agradecer a mi familia el apoyo dado en todo momento y la ayuda que siempre me ha ofrecido. A mi pareja, por todas las horas que hemos pasado juntos en el escritorio estudiando. A mi tutor, por brindarme la oportunidad de desarrollar este proyecto y acompañarme en el proceso. Por último a mis mascotas por aguantarme durante todo este tiempo.

INDICE DE CONTENIDOS

INDICE DE FIGURAS

1 Introducción

1.1 Motivación

La motivación principal para la realización de este proyecto es preservar el derecho a la información real que tienen las personas.

Dado el crecimiento de las tecnologías en las últimas décadas y en especial al nacimiento de las redes sociales, la información está mas al alcance de la mano que nunca, pero. Esto es beneficioso para todo el mundo pero el exceso de información hace difícil reconocer la verdad y es aquí donde de generan las fake news o noticias falsas.

El carácter principal de las fake news es desinformar o crear confusión en el lector consiguiendo. Dado que es relativamente fácil de generar y gracias a la creación de Internet, redes sociales y el acceso a audiencias masivas es una práctica muy extendida, tanto es así que existen empresas dedicadas a generar noticias falsas.

De esta mala práctica llevada a cabo con la información nace este proyecto que puede servir de guía para determinar la calidad de la información a la que tenemos acceso, de momento mediante los tweets de Twitter, pero podría extenderse a cualquier medio de comunicación existente hoy día.

1.2 Objetivos

El objetivo de este proyecto, es desarrollar un cliente de Twitter extendido y parte del servidor, que sirva como guía para determinar si la información contenida en el texto de una tweet puede contener o no información engañosa o falsa.

Esta aplicación que tiene como nombre Fake tweet hunter, mostrará los tweets del usuario que se dé de alta indicando el porcentaje de credibilidad que tiene cada tweet almacenado en la aplicación.

Como se trabaja con tweets uno de los objetivos principales es la integración de la API de Twitter dentro de la aplicación para su correcto funcionamiento tanto para recepción de tweets como para poder realizar el inicio de sesión con las propias credenciales de Twitter.

1.3 Organización de la memoria

Esta memoria se ha estructurado de la siguiente manera:

- **Sección 1:** En esta sección se realiza una introducción al proyecto para dar una idea al lector del contenido del trabajo, así como mostrar la estructura del propio documento.
- **Sección 2:** En esta sección se detalla el estado del arte, donde se explican las fake news, las aplicaciones móviles y las posibles alternativas.

- **Sección 3:** En esta sección se explican las diferentes tecnologías utilizadas, las funcionalidades que tendrá el producto final, así como una explicación del diseño de las diferentes partes.
- **Sección 4:** En esta sección se realiza una explicación detallada del proceso de desarrollo del proyecto tanto por parte de la aplicación como por parte del servidor.
- **Sección 5:** En esta sección se detallan las diferentes pruebas realizadas durante y al finalizar el proyecto.
- **Sección 6:** En esta última sección se encuentran las conclusiones extraídas del desarrollo de este proyecto y las posibles líneas futuras de la aplicación.

2 Estado del arte

2.1 Fake news

El termino fake news procede del inglés y son un producto periodístico, o simula serlo, que se difunden mediante Internet, prensa escrita o cualquier otro media de comunicación para desinformar de manera deliberada y engañar.
El uso de fake news o noticias falsas se ha incrementado en los medios de comunicación en los últimos años, debido en parte, al crecimiento del uso de las redes sociales y de Internet como medio para buscar información.
Aunque sea un término que se utiliza últimamente con más frecuencia, no es nuevo, ya se utilizaba en la antigüedad. Y en la actualidad es utilizado con más intensidad en temas políticos, sobretodo en pre-campañas electorales.

Las fake news suelen iniciarse precisamente en las redes sociales ya que es más fácil de propagarlas y en segundos pueden extenderse a nivel mundial, incluso a veces llegar a los medios de comunicación.

Un ejemplo de esto se puede ver durante las elecciones de EEUU en el 2016 (Figura 1) donde en el mes de las elecciones se incrementa en mas de 5 millones las noticias que se comparten, comentan o tienen algún tipo de reacción [10].

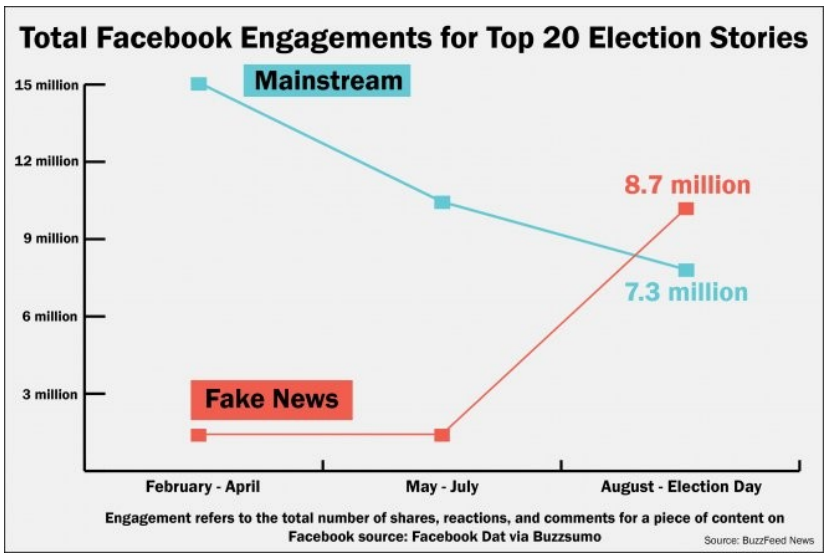

Figura 1: Grafica de fake news en las elecciones de EEUU en 2016

2.2 Aplicaciones móviles

Una aplicación móvil es una aplicación informática diseñada y desarrollada para ejecutarse en teléfonos móviles inteligentes. Actualmente existen muchas y de diversas categorías que permiten al usuario realizar tareas de todo tipo en cualquier ámbito, desde educación, profesional, financieras, acceso a servicios o instituciones, etc.

Las aplicaciones se pueden encontrar dentro de las plataformas de las compañías propietarios como IOS, Android (las más conocidas y numerosas) o Blackberry, Windows Phone y otras (menos conocidas y numerosas).
Se pueden encontrar aplicaciones gratuitas totalmente, gratuitas con posibilidad de compras dentro de la propia aplicación o de directamente de pago [6].

2.2.1 Breve historia

Con la llegada de los móviles de tercera generación en el 2007 llegaron las aplicaciones móviles de la mano de Apple y Google, dos de las principales *marketplace*, cuando un año más tarde en 2008 lanzaron sus respectivas tiendas de aplicaciones App Store y Google Play (inicialmente llamada Android Market).

Figura 2: Logo App Store de Ios

Figura 3: Logo Google Play de Android

Desde su nacimiento, el crecimiento de estas dos tiendas ha experimentado una subida muy considerable, estableciendo el número actual de aplicaciones por cada tienda en más de 3.750.000 en el caso de Google play y de más de 2.100.000 en el caso de App store. La evolución del número de aplicaciones se puede observar en la gráfica (Figura 4) donde se puede ver un crecimiento continuo y en los dos últimos años un incremento considerable en la App Store de IOS, mientras que para Google Play el incremento es continuo en toda su existencia (Figura 5).

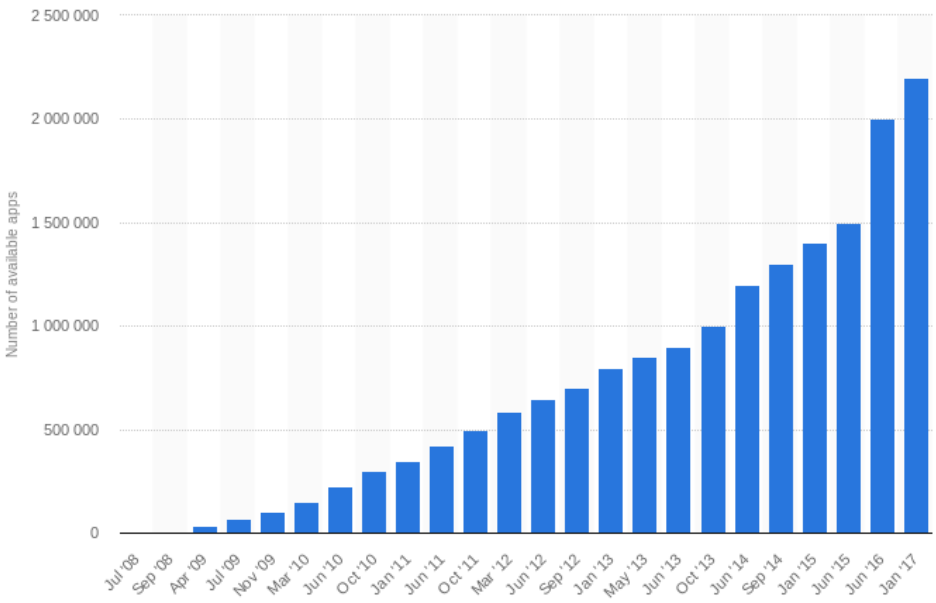

Figura 4: Gráfico de la evolución del número de aplicaciones en App Store

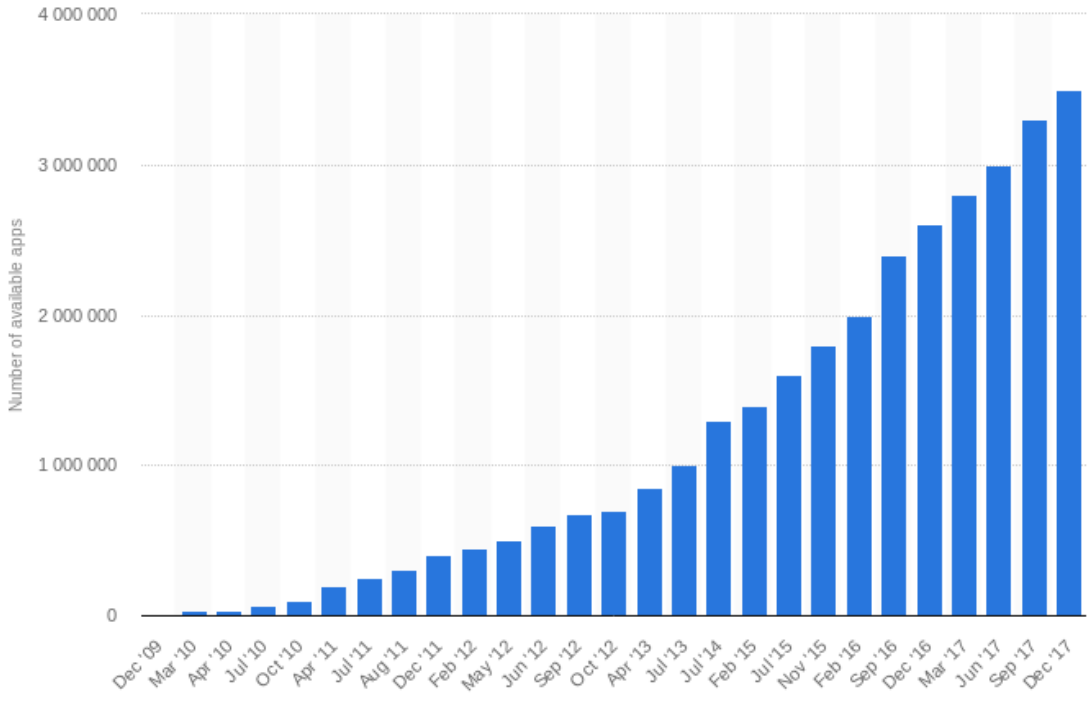

Figura 5: Gráfico de la evolución del número de aplicaciones en Google Play

Paradójicamente en el caso de las descargas aunque App Store disponga de menos aplicaciones gana con diferencia a Google Play como se puede observar en las gráficas, contabilizando estas descargas en billones [8].

En el caso de App store las descargas a fecha de Diciembre de 2017 se estima en unos 180 Billones.

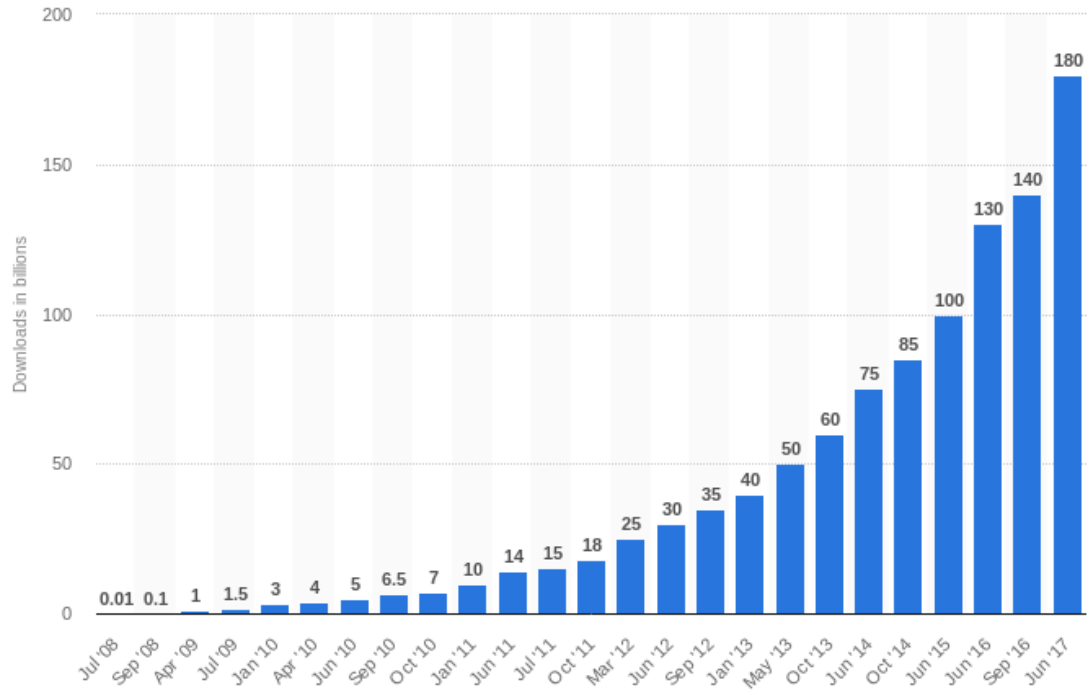

Figura 6: Número de descargas, en billones, de App Store

En el caso de Google Play las descargas a fecha de Junio de 2017 se estiman en 65 Billones.

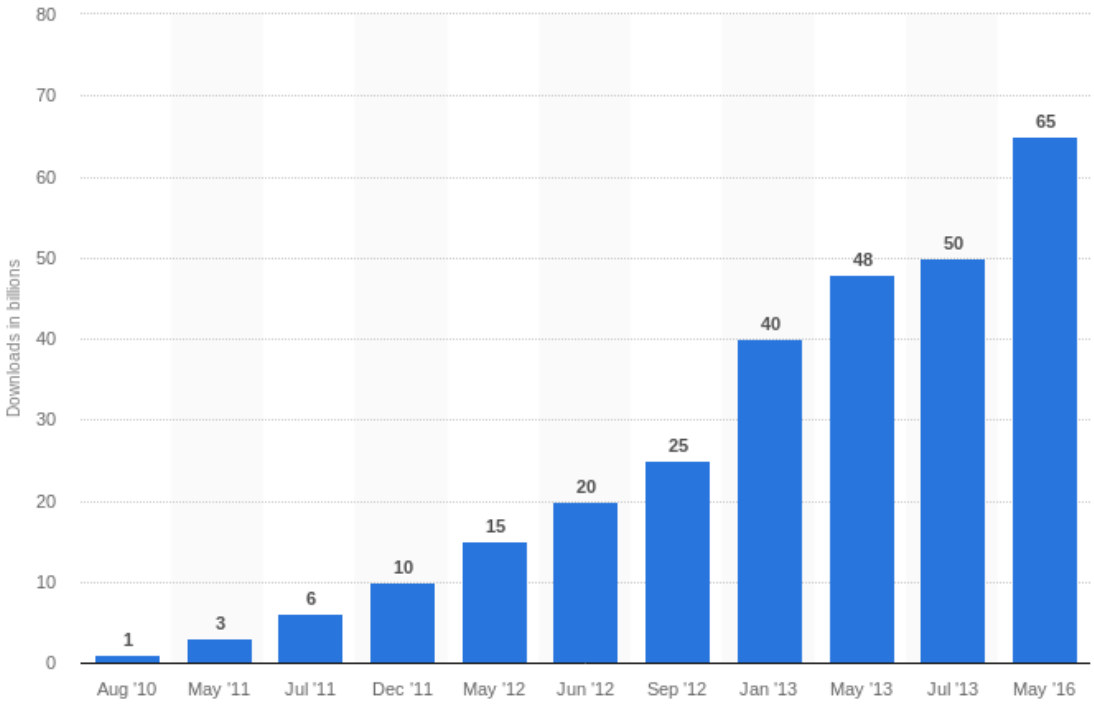

Figura 7: Número de descargas, en billones, de Google Play

2.2.2 Tipos de aplicaciones móviles

Este incremento viene dado por la facilidad de acceso a un teléfono móvil que se tiene en la actualidad, pero también por que cada vez existen más maneras de crear estas aplicaciones [9].

Entre ellas están las siguientes:

Nativas
Son aplicaciones que se desarrollan específicamente para cada sistema operativo, con su propio lenguaje de programación, Java para Android, Objective-C para IOS.
Como ventajas se puede destacar que al ser del propio sistema operativo se tiene acceso prácticamente a la totalidad de los componentes del móvil y todas sus funcionalidades.
Como principal desventajas es que son más costosas de desarrollar por la complicación del lenguaje, que es más especifico. Este tipo de aplicaciones no requieren conexión a Internet.

Web apps
Por otro lado existe un tipo de "aplicaciones" llamadas Web apps, realmente no se pueden considerar aplicaciones como tal ya que se desarrollan como si fueran una página web pero realizando una adaptación a pantallas de teléfono móvil.
Como ventajas tienen que son mucho más económicas y rápidas de hacer.
Como principal desventajas es que prácticamente se trata de una página web por tanto no se tiene acceso a los componentes de teléfono móvil y que requiere conexión a Internet para funcionar.

Apps híbridas
Por último existen las llamadas Apps híbridas, que es el que se utiliza en este proyecto. Se trata de una mezcla de las dos anteriores, tienen la facilidad de la creación y permiten el acceso a las funcionalidades y componentes del teléfono móvil.
Como ventajas tienen que se desarrollan con lenguaje de programación web, es decir HTML, JavaScript y CSS, con lo cual facilita muchísimo el desarrollo y es más económico, también que permiten el acceso a las funcionalidades del dispositivo.
Cómo principal desventajas tienen que, aunque si tengan acceso a estas funcionalidades del dispositivo, no es 100% accesible, existen componentes a los que no se tienen acceso.

2.3 Otras aplicaciones que combaten las fake news

En esta sección se detallarán las alternativas existentes con la misma temática que la de este proyecto, con el objetivo de ver cómo funcionan y tratar de ofrecer una aplicación al menos a la altura de estas alternativas.

Fakebuster: Duck Vs Fake News
Esta aplicación está disponible tanto para IOS como para Android, se trata de un juego destinado a las fake news. La dinámica es muy sencilla, va mostrando noticias y hay que indicar si son verdad o mentira, una vez seleccionada una opción, muestra si esa noticia es real o no. Cuenta con un gran número de noticias actuales y con cada versión añaden nuevas. Aunque no es una aplicación que le puedas exponer una noticia externa para procesarla, se podría considerar una introducción a las fake news ya que las

noticias que muestran están contrastadas y se puede utilizar para saber si los titulares que ves son fake news o no.

Figura 8: Logo de Fakebuster: Duck Vs Fake News

Vozz

Esta aplicación se puede encontrar tanto en IOS como en Android. Desarrollado por un equipo venezolano-panameño, cuenta con un equipo de periodistas y desarrolladores que contrastan y confirman las noticias para ofrecerlas al usuario de la aplicación con garantías de que lo que están leyendo es real.

Su funcionamiento se basa en mostrar las noticias más recientes que se comparten por redes sociales, filtrarlas y una vez que su equipo las ha contrastado y verificado, son distribuidas en diferentes categorías.

Cuenta con cobertura en Venezuela, Panamá, Colombia y una sección de edición internacional y actualmente cuenta con más de 10.000 descargas.

Figura 9: Logo de Vozz

Facebook

Ya es conocido que Facebook es una de las redes sociales más utilizadas y donde se comparten noticias a diario. Dada la gran cantidad de noticias que se puede llegar a publicar en un día en esta red social es muy fácil que muchas sean fake news y recientemente Facebook a dado a conocer que esta implementando mecanismos para evitarlas. Ha empezando desarrollando una herramienta para que los usuarios obtengan información y consejos para identificarlas [3]. Entre las opciones que ofrece están, revisar la URL del sitio que está publicando el artículo, poder investigar las fuentes,

comparar la información con otros reportes del mismo tema así como ver cuanta gente comparte un artículo y que tipo de personas lo hacen.

Figura 10: Logo de Facebook

Google

Al igual que Facebook, Google también está dispuesta a combatir las fake news y para ello recientemente ha publicado que invertirá 300 millones de dolares en un nuevo proyecto llamado News Initiative cuyos objetivos son luchar contra las fake news, especialmente en el ámbito de la política, y prestar apoyo a compañías editoras de medios. El proyecto, que se irá desarrollando durante los próximos tres años, contará con un proceso de ventas de suscripciones, promoción y fuentes de noticias "fidedignas" en los resultados de búsquedas [4].

También ha renovado sus plataformas para darle prioridad al periodismo de calidad y dar mejor posicionamiento a los medios que ofrezcan información veraz.

Figura 11: Logo de Google

Twitter

Otra red social que se suma a la guerra contra las fake news es la propia plataforma que se utiliza en este proyecto. Twitter al igual que Facebook es también una de las más utilizadas a nivel mundial para compartir noticias de manera rápida y a implementado una serie de medidas para intentar evitar que las noticias falsas se propaguen [2]. No permitirá que los usuarios de esta red social publiquen mensajes idénticos desde varias cuentas. Tampoco permitirá que los usuarios utilicen software para realizar varias acciones al mismo tiempo, como dar me gusta o retuitear mensajes desde varias cuentas. Estas acciones intentan combatir a los bots que generar tweet

automáticamente, los mismos bots que investigadores y autoridades estadounidenses dicen que diseminaron propaganda antes de las elecciones presidenciales del 2016.

Figura 12: Logo de Twitter

3 Análisis y Diseño

3.1 Análisis de requisitos

En esta sección se detallarán los requisitos funcionales y no funcionales a cumplir por el sistema.

3.1.1 Requisitos funcionales

En este apartado se listarán los requisitos funcionales, centrados en el funcionamiento del sistema para usuarios de la app o web.

RF 01: Tanto el registro como el acceso se realizará mediante Oauth a través de Twitter y con las mismas credenciales de Twitter.
RF 02: En el registro el usuario dispondrá de 100 tweets publicados o retuiteados por él y de 100 tweets de la página principal de su cuenta.
RF 03: El usuario dispondrá de información sobre la veracidad de la información en cada tweet listado dentro de la aplicación.
RF 04: El usuario dispondrá de los tweets creados y retuiteados por él en una pestaña a parte de los tweets de la página principal de su cuenta.
RF 05: El usuario podrá cerrar sesión en cualquier momento.
RF 06: El sistema guardará información del usuario devuelta por Twitter

3.1.2 Requisitos no funcionales

En este apartado se listarán los requisitos no funcionales, centrados en el rendimiento del sistema.

RNF 01: La aplicación se podrá instalar en dispositivos Android y IOS.
RNF 02: El sistema será accesible en formato web mediante navegador.
RNF 03: El usuario debería ingresar en la aplicación mediante credenciales de Twitter.
RNF 04: El sistema deberá guardar los tweets obtenidos de Twitter para ser lo más eficiente posible.
RNF 05: El sistema deberá ser fácil de utilizar.
RNF 06: El sistema deberá almacenar tweets sin repetirlos.

3.2 Tecnologías utilizadas

Meteor
Meteor es la base de este proyecto, es lo que hace posible la creación de esta aplicación híbrida disponible para Android, IOS y web [11]. Es un proyecto open source y funciona sobre Linux, Mac y Windows.
Se podría definir como un framework full-stack, pero es más acertado definirlo como una plataforma para desarrollar aplicaciones web, el motivo de esto es que no es propiamente un framework si no que realiza la función de unir diferentes framework para que coexistan entre sí.

Figura 13: Logo de Meteor

Se distinguen dos partes, cliente y servidor, diferenciadas entre si, ya que existe código que se ejecuta sólo en el servidor y código que se ejecuta sólo en el cliente. La parte del servidor funciona sobre NodeJS, que se verá más adelante. La parte del cliente funciona mediante HTML, JavaScript mayoritariamente y CSS.
Para garantizar la seguridad solo el código en el servidor tiene acceso a la base de datos.
Como se puede observar en el esquema que Meteor propone (Figura 14) existen varias capas distribuidas de la siguiente manera:

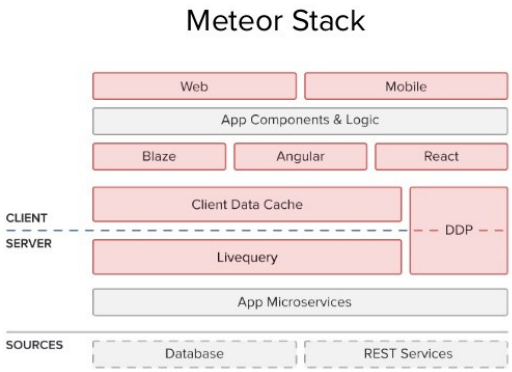

Figura 14: Esquema de arquitectura que propone Meteor

- **Capa 1**: Web y Mobile, permite la creación tanto de aplicaciones para móvil mediante Cordova, como páginas web para navegadores.
- **Capa 2**: Componentes y lógica, todo el funcionamiento de botones, formularios, etc., así como los componentes detrás de la parte visual.
- **Capa 3**: Blace, Angular o React, es posible seleccionar entre estas tres librerías que son las que se encargan de renderizar el código para generar la aplicación.
- **Capa 4**: Caché cliente, Livequery y DDP, por parte del cliente es donde se encuentra la BBDD del cliente, por parte del servidor se encuentra Livequery, tecnología de Meteor encargada de conectar la base de datos con el DDP mediante websockets.
- **Capa 5**: Microservicios, pensado para que alojar microservicios con los que se va conectando la aplicación y son completamente independientes.
- **Capa 6**: Base de datos, Servicios REST.

Lo que aporta de nuevo Meteor sería la capa numero 4, la caché del cliente, el livequery en el servidor y el DDP, protocolo de comunicación entre cliente y servidor a través de pequeños mensajes de manera bidireccional, el resto son distintas tecnologías que Meteor une para que funcionen a la vez.

Blaze

Blaze es una poderosa biblioteca para crear interfaces de usuario escribiendo plantillas HTML reactivas. En comparación con el uso de una combinación de plantillas tradicionales y jQuery, Blaze elimina la necesidad de toda la "lógica de actualización" en su aplicación que escucha los cambios de datos y manipula el DOM. En cambio, las directivas de plantillas familiares como {{#if}} y {{#each}} se integran con la "reactividad transparente" de Tracker y los cursores de la base de datos de Minimongo para que el DOM se actualice automáticamente.

Figura 15: Logo de Blaze

NodeJS

Creado en 2009 ha supuesto muchos cambios en el desarrollo web. Desarrollar una aplicación cliente/servidor con Node.js supone que ahora la parte del servidor también va a estar programada en JavaScript.

Figura 16: Logo de NodeJS

Es de código abierto, gratuito y multiplataforma. Es donde radica la principal diferencia en base a arquitecturas de aplicaciones anteriores ya que donde antes, el cliente hacía una petición al servidor y quedaba esperando una respuesta, nodeJS una vez enviada la petición esta disponible para realizar más peticiones y a medida que el servidor va devolviendo respuestas las va tratando. Esto es posible gracias a la programación asíncrona de un subproceso, sin bloqueos, que es muy eficiente desde el punto de vista de la memoria.

MongoDB

Se trata de una base de datos no relacional, no se basa en tablas con relaciones como las bases de datos relacionales como SQL, si no que está orientada a documentos.

Proporciona más potencia para aplicaciones móviles y web y es más flexible que una base de datos relacional. Este tipo de bases de datos admiten diferentes formatos de datos extraídos del documento en sí, en lugar del formato limitado de las relacionales. Es el estándar en Meteor.

Figura 17: Logo MongoDB

Como se ha visto en el esquema de la arquitectura, en la parte del cliente también existe una base de datos MongoDB, llamada Minimongo.
Minimongo es una versión del lado del cliente de MongoDB que actúa principalmente como un emulador de MongoDB. Obtiene muchas características de MongoDB y trabaja con la misma API, por lo tanto, no es necesario preocuparse por muchos ajustes de desarrollo. Puede almacenar la información necesaria del lado del cliente y obtener el potencial para crear un código portátil. El paquete Minimongo permite consultar en vivo la base de datos para actualizaciones en tiempo real. Si los datos generados por la base de datos del lado del cliente no coinciden con los del lado del servidor, los datos se actualizan rápidamente.

3.3 Funcionalidades

En esta sección se mostrarán de una manera gráfica y breve la funcionalidad de la aplicación.

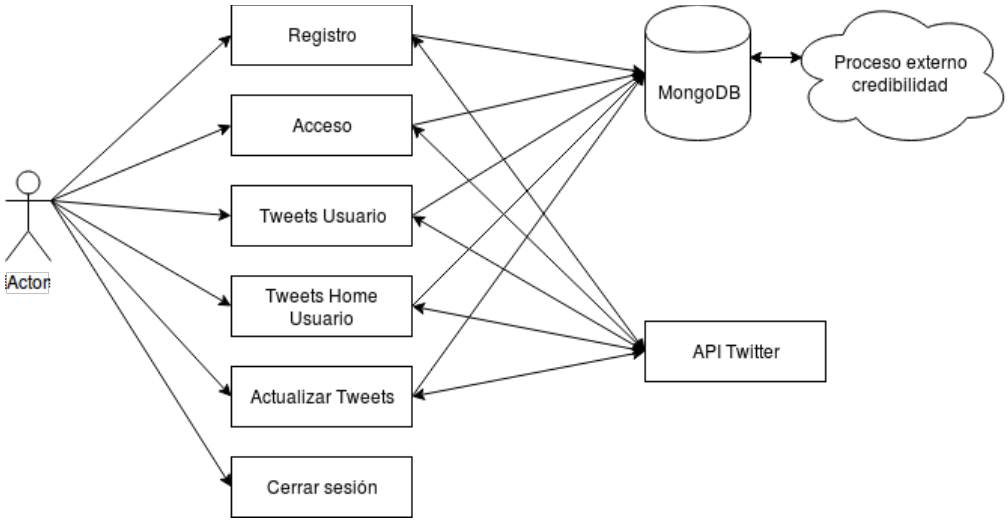

Figura 18: Diagrama funcionamiento Fake tweet hunter

En este diagrama se puede observar las acciones que el usuario puede realizar sobre la aplicación y como cada proceso interactúa con otros procesos ajenos al usuario.

3.4 Diseño APP y PC

En esta sección veremos visualmente como es la aplicación final tanto en móvil como en versión web.

La manera en la que se ha distribuido la estructura del proyecto a nivel de archivos viene marcada por Meteor, la parte del cliente está bajo la carpeta client, la parte del servidor se encuentra bajo la carpeta server. Con esto se diferencia cada parte y se puede extrapolar a cualquier parte, es decir, si se crea otra carpeta y dentro pones esta estructura de cliente y servidor, la parte que cuelga de cliente sólo será visible para el cliente y lo mismo con el servidor (Figura 18). A parte también existen otras carpetas establecidas por Meteor:

- ***Public**:* todos los archivos de esta carpeta son enviados al cliente tal cual, de forma que podamos referenciarlos en nuestro HTML, CSS
- ***Private***: Todos los archivos de esta carpeta son solo accesibles desde código del servidor y se cargan utilizando la API de Assets
- ***node_modules***: La carpeta donde se guardan los paquetes npm que podremos cargar desde Meteor directamente.
- ***Imports***: Para crear nuestros propios módulos debemos incluirlos en la carpeta imports. La razón por la que Meteor tiene una carpeta especial para los módulos es no romper compatibilidad con aplicaciones ya existentes.

Figura 19: Estructura de carpetas del proyecto

Por otro lado tenemos la estructura de diseño donde se aprecia un *Main* que es la base de la estructura y contiene el menú y el fondo donde se van intercambiando las páginas en función de lo seleccionado en el menú.

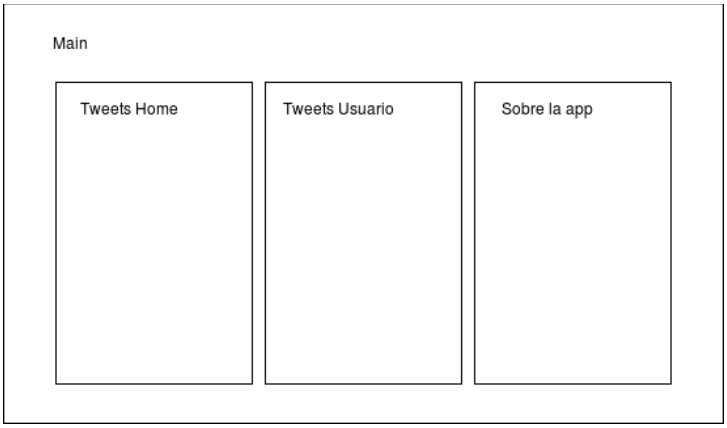

Figura 20: Estructura visual del proyecto

En versión web el aspecto es el siguiente, la pantalla principal muestra un menú con una opción de inicio y otra “sobre la APP” que muestra información sobre la aplicación.

Al hacer clic en conecta con Twitter, muestra una ventana nueva con el acceso a Twitter para introducir las credenciales de la cuenta de Twitter del usuario para permitir el acceso a la aplicación. Esta acción sólo es necesario realizarla una vez, en futuras conexiones esta acción es recordada.

La acción de registro y de inicio de sesión se realizan mediante el mismo botón.
Es de manera interna como se detecta si existe cuenta de esas credenciales de Twitter para crear usuario o permitir el acceso directamente.

Figura 21: Diseño visual de la Pantalla principal

Figura 22: Registro y acceso a la aplicación mediante Twitter

Una vez realizado el acceso, se muestra la lista de tweets de la home de Twitter, donde se puede aprecias el usuario que escribe el tweet o al que se realiza el retuit, seguido del texto del propio tweet y a la izquierda de cada tweet el porcentaje de credibilidad del tweet, en caso de que esté calculado, si no muestra un guión.

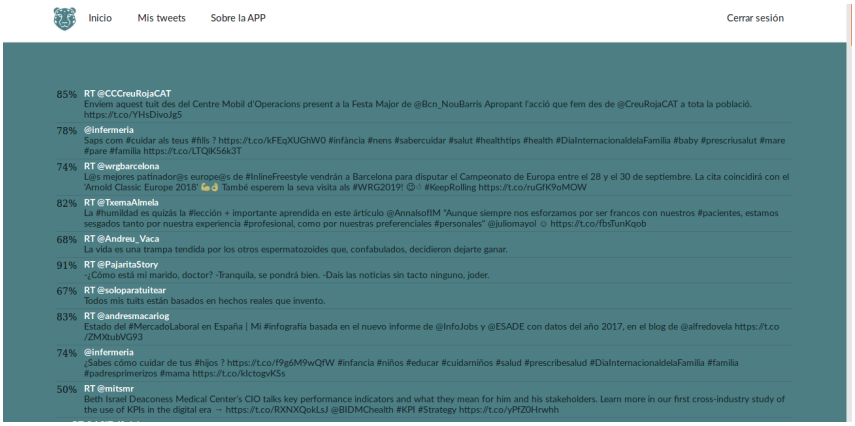

Figura 23: Diseño visual pantalla principal después del acceso

Figura 24: Diseño visual pantalla "Mis Tweets"

En el caso de la aplicación la información es la misma como se puede observar

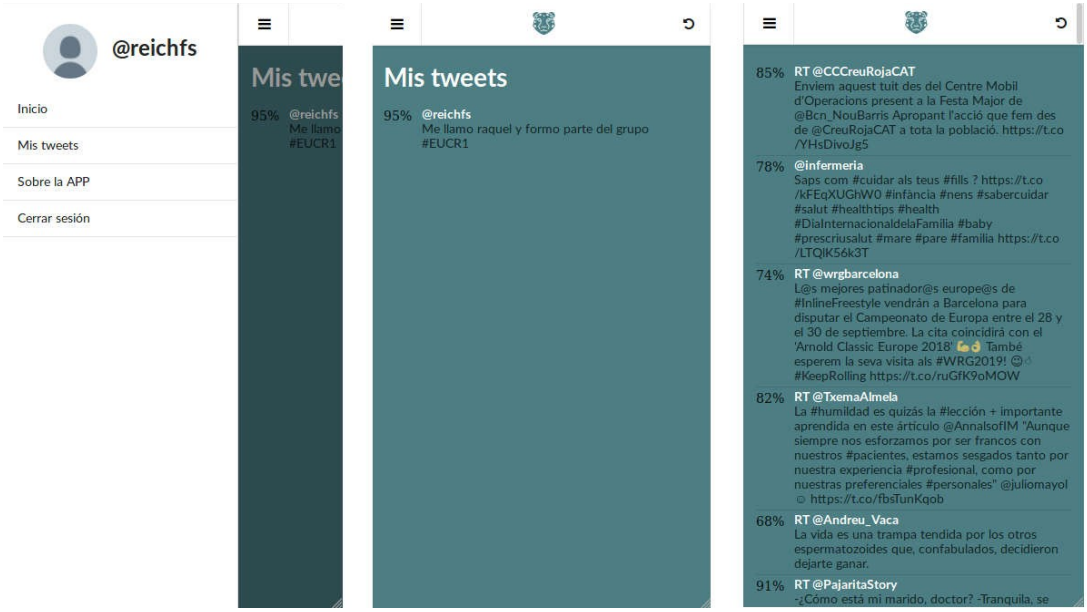

Figura 25: Diseño App. Menú (Izquierda), Pantalla Mis Tweets (Central), Pantalla tweets home (Derecha)

3.5 Diseño BBDD

El diseño de la base de datos es muy sencillo, como se ha explicado anteriormente se basa en colecciones, cada colecciones que se crea viene identificado con un id propio que genera Mongo de manera automática.

Esta aplicación consta de 3 colecciones:

Users

En esta colección se guarda información sobre el usuario, a parte del _id que genera el propio MongoDB, se guarda la fecha de creación, bajo services se guarda información que devuelve Twitter al realizar el registro como tokens de acceso, imágenes de perfil, nombre de usuario del perfil, id de Twitter y bajo profile se guarda en este caso el nombre real de la persona que tiene en Twitter.

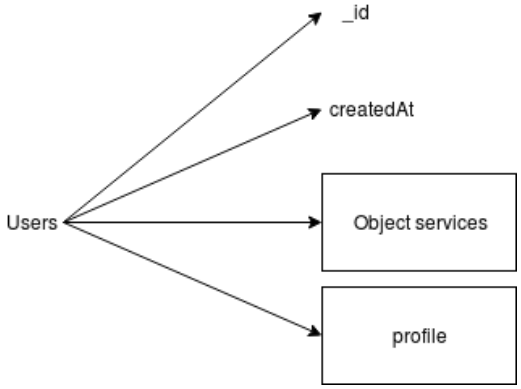

Figura 26: Diseño de documento Users en MongoDB

Tweets
En esta colección se guarda información sobre los tweets que se van descargando de Twitter. A parte del _id que genera el propio MongoDB se guarda un objeto tweet que es la información completa que devuelve Twitter sobre cada tweet. Esta información contiene campos como el id del tweet, fecha de creación, texto, información del usuario que lo ha creado, información sobre multimedia que pueda contener el tweet como imágenes o videos entre otra información. A parte también se guarda un campo reliability con el porcentaje de credibilidad del tweet.

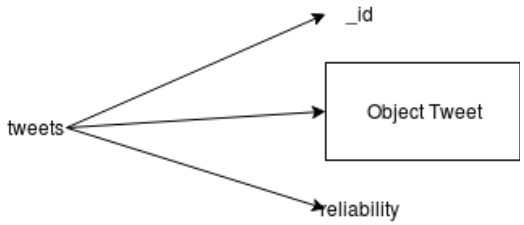

Figura 27: Diseño de documento Tweets en MongoDB

UsersTweets
En esta colección se guarda información sobre la relación que tiene los usuarios con algunos tweets. Esto es debido a que en la página principal de Twitter aparecen tweets de personas a las que el usuario sigue, estos tweets contienen información del creador del tweet o de la persona que realiza el retuit, pero no contiene ninguna relación con el usuario de la cuenta, por eso es necesario crear esta tabla donde se guarda, a parte del _id que genera el propio MongoDB, el ID del usuario de la cuenta y una lista con todos los tweets que se van descargando de su página de inicio en Twitter, en esta lista no se incluyen los tweets que el usuario crea o retuitea ya que estos tweets si que estan identificados con el id del usuario y se pueden recuperar de manera directa.

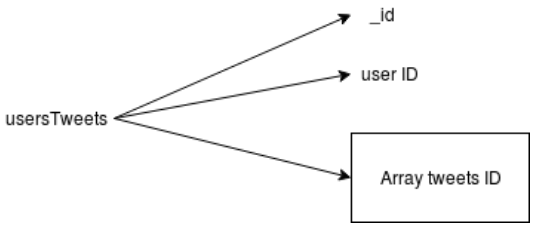

Figura 28: Diseño de documento UsersTweets en MongoDB

4 Desarrollo

En esta sección se detallará el proceso de desarrollo que se ha seguido en este proyecto.

Se pueden distinguir dos partes diferenciadas, el desarrollo de la aplicación, que incluye la aplicación móvil y la plataforma web por parte del cliente y por parte del servidor, y el desarrollo del servidor, que incluye el despliegue en el servidor.

Meteor dispone de paquetes que añaden funcionalidades extra al proyecto. Estos paquetes están disponibles a través de Atmosphere. Atmosphere es el catálogo de paquetes, recursos y herramientas de Meteor. Para realizar la instalación de estos paquetes, siempre hay que seguir la misma estructura *meteor install <nombre del paquete>*. Los que se han utilizado en esta aplicación son:

SemanticUI
SemantiUI es un marco de desarrollo que ayuda a crear diseños agradables de manera fácil e intuitiva utilizando HTML y clases predefinidas que tienen asociado un estilo mediante CSS ya establecido. Aunque estos estilos estás definidos, se pueden modificar y adaptar a las necesidades que se requieran.
En este caso, el paquete SemanticUI requiere de otro paquete para funcionar *juliancwirko:postcss*, este paquete utiliza LESS en vez de CSS para crear toda la estructura de estilos predefinida.

Una vez instalado los paquetes hay que hacer una pequeña configuración en un archivo de postcss para establecer los estilos que se quieren utilizar ya que hay varios para elegir, así se evita tener que cargarlos todos y sólo cargas el que vas a utilizar.
Esta configuración se realiza en un archivo llamado *custom.semantic.json* que hay que crearlo en una carpeta especifica *client/lib/semantic-ui*

```
{
"definitions": {
  "accordion" : true,
  "ad"        : true,
  "api"       : true,
  /* etc */
},
"themes": {
  "amazon"    : false,
  "basic"     : false,
  "bookish"   : false,
  "bootstrap3" : false,
  "chubby"    : false,
  "classic"   : false,
  "default"   : true,
  /* etc */
}
```

Figura 29: Configuración del paquete SemanticUI

Como se puede apreciar bajo el nodo *"themes"* existen varios tipos de temas y el único que esta a *true* es el *"default"* que será el que cargará para su utilización.
Una vez hecho esto, se genera toda la estructura LESS y está lista para su utilización.

Accounts-twitter
Este paquete se ha utilizado para poder realizar el registro y el acceso a través de la API de Twitter de manera rápida ya que tiene la función implementada y en conjunción con el siguiente paquete, que es requisito para su funcionamiento, se puede realizar la acción de registro y acceso al sistema.

Service-configuration
Este paquete es requisito del anterior, su función es sencilla, sirve para configurar las *key* del consumidor de Twitter necesarias para poder conectarse con la API de Twitter.

mystor:device-detection
Este paquete se ha utilizado para poder distinguir entre si lo que está accediendo al sistema es un dispositivo móvil o un ordenador. Su función principal ha sido poder diferenciar las partes de código que son distintas entre móvil y ordenador, como por ejemplo el menú.

iron:router
Este paquete es bastante importante ya que funciona sobre el principio " *data on the wire* ". El servidor no utiliza rutas ni HTML, es la aplicación en cliente la encargada de renderizar el HTML obteniendo los datos a través del protocolo DDP.

twit npm
El paquete más importante, es el que se encarga de comunicarse con la API de Twitter para realizar las llamadas oportunas, en las siguientes secciones se detalla su utilización.

4.1 Desarrollo de la aplicación, cliente

Como se ha explicado anteriormente Meteor diferencia la parte del cliente y la del servidor en el código y el cliente únicamente tiene acceso a la parte del cliente, en ningún momento puede acceder al código de la parte del servidor, solamente a través de llamadas a funciones.

En la parte del cliente se encuentran 2 tipos de archivos, los que controlan la parte visual, hecha con HTML y CSS y la que controla la conexión con la parte del servidor, hecho en JavaScript.

Tanto la aplicación móvil como la web comparten código con algunas excepciones que se irán remarcando a medida que vaya avanzando la explicación.

Meteor establece una serie de patrones para su utilización, en este proyecto se utiliza Blaze que funciona mediante Spacebars *{{}}*, con la doble llave se tiene acceso a variables definidas en la parte JavaScript del cliente. Todos los template de blaze se declaran utilizando la etiqueta `<template>` y todo lo que vemos dentro son helpers de Blaze.

Por otro lado se utiliza un sistema de publicaciones de Meteor, con este método el cliente sólo dispone de lo que necesita y nada más, para eso se utiliza la caché en la parte del cliente que contiene Minimongo.

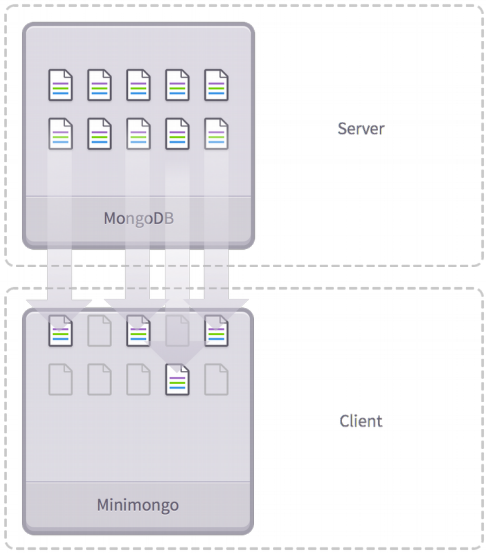

Figura 30: Esquema distribución bases de datos en Meteor

Desde la parte del servidor se realizan publicaciones con parte de la base de datos que se va a utilizar en el cliente, lo que hace Meteor es mantener sincronizada esa parte en el cliente mediante suscripciones, así el cliente tiene la información al instante, sin esperas y lo más importante, sólo tiene la información que necesita, no tiene acceso a otra información.

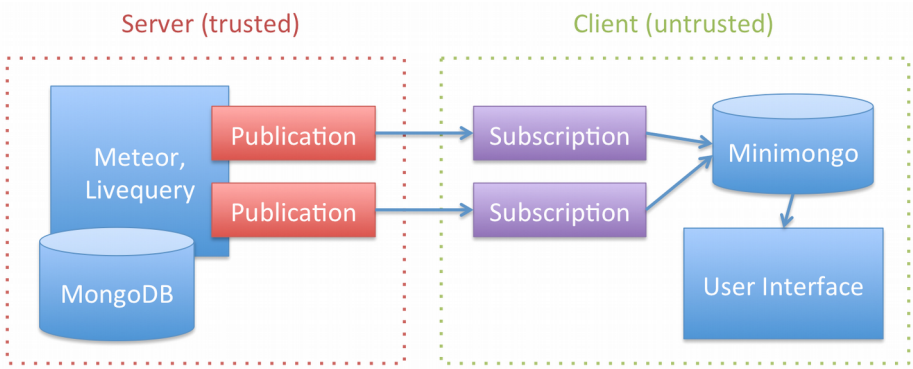

***Figura 31**: Funcionamiento de publish & suscribe en Meteor*

A continuación se detalla cada parte de la aplicación.

Main

En la parte estética se establece el menú en función de si es para dispositivo móvil o para web mediante el paquete comentado con anterioridad y las etiquetas *{{#if isPhone}}* y *{{#if isDesktop}}*.

Tambien se controla que opciones mostrar en el menú en función de si un usuario esta con una sesión iniciada o no mediante la etiqueta *{{#if currentUser}}*.

Por último se utiliza la etiqueta *{{> yield}}*, esta etiqueta es una etiqueta comodín ya que la utiliza el paquete iron:router para saber donde tiene que sustituir los templates que se le

configuren ya que como se ha explicado en la sección Diseño, esta parte *Main* es la base desde donde se irán intercambiando las páginas en función de la opción del menú seleccionada.

En la parte JavaScript se establecen los imports de los paquetes que se utilicen. También se establecen los helpers, methods y en este caso la configuración de iron:router.

Los helpers se utilizan pasar información al template, lo que se defina aquí, se puede recuperar en el template mediante las dobles llaves {{}}.

Los events son eventos que se generan el el template como clics en botones, en *Main* se han establecido 2 events uno para el actualizar los tweets y otro para cerrar la sesión. La estructura es la siguiente:

```
Template.main.events({
  'click .signOut' : function() {
    Meteor.logout();
    Router.go('/');
  },
  'click .buttonSync' : function() {
    Meteor.call('actualizaTweetsUser', Meteor.user().services.twitter.screenName);
  }
});
```

Figura 32: Eventos creados en Main

En el caso de cerrar sesion, se realiza un logout(), función propia del paquete Accounts-twitter.
En el caso del segundo evento, realiza una llamada a una función en la parte del servidor que actualiza los tweets.

Por otro lado está configurado iron:router. Mediante *Route.configure*, se establece que la base es este template *Main*

Una vez establecida la base, a cada ruta que se le defina, se sustituirá en la etiqueta yield establecida en el template.

```
Router.route('/misTweets', {
    name: 'Mis Tweets',
    template: 'misTweets',
    subscriptions: function() {
        this.subscribe('tweets');
        this.subscribe('users.user');
    },
    action: function() {
        this.render();
    },
    after: function() {
        document.title = 'Fake Tweets Hunter - Mis Tweets'
    }
});
```

Figura 33: Ejemplo de route

También se establece las suscripciones descritas con anterioridad, en este caso se tiene acceso a los tweets del propio usuario y a información del usuario y nada más. Otras configuraciones son el template que se va a utilizar en esta ruta, el titulo que se le asignará

y la url que se le asignará. Las otras secciones de la aplicación se establecen de la misma manera.

Registro / Acceso
En la parte estética es muy sencilla, sólo existe un botón que se utilizará para realizar el registro y el acceso al sistema.

En la parte JavaScript sólo existe un evento que detecta el clic en el botón y realiza un loginWithTwitter, función propia del paquete Accounts-twitter. Esta función conecta con Twitter mediante Oauth y las credenciales del usuario a través de la ventana que se abre. Este paquete de manera automática detecta si el usuario existe o no para crear el usuario o permitirle el acceso.

Una vez comprobado el usuario si existe, realiza 2 llamadas a funciones de la parte del servidor.

Tweets de la Home
En la parte estética tenemos una comprobación para saber si existe login realizado.
También se muestra la lista de tweets de la Home de Twitter del usuario mediante el iterador *{{#each tw in getInfoTweet}}* donde getInfoTweet es una variable declarada en el helper que contiene una lista de los tweets devueltos desde el servidor.

En la parte JavaScript se han definido 5 helpers. Los helpers utilizan el contexto del template y datos de otras fuentes, como puede ser una colección de la base de datos, para pasar información al template. En este caso devuelven la lista de tweets con una llamada a la base de datos Minimongo en la parte del servidor mediante *Tweets.find({})*. Tweets es la variable definida para el acceso a la colección de la base de datos mediante *Tweets = new Meteor.Collection("tweets");*

```
Template.listaTweets.helpers({
  'getInfoTweet': function() {
    return Tweets.find({});
  },
  'isPhoto': function(medio) {
    return medio.type === "photo";
  },
  'isVideo': function(medio) {
    return medio.type === "video";
  },
  'ismp4': function(medio) {
    return medio.content_type === "video/mp4";
  },
  'percentReli': function(res) {
    if (res == 'pending'){
      return '-';
    }
    return res*100 + '%';
  }
});
```

Figura 34: Ejemplo de helpers

Los otros helpers definidos se utilizan para saber información del tweet.

El helper *percentReli* es llamado desde el template para cada tweet mediante *{{percentReli tw.reliability}}*, tw.reliability es lo que se recibe por cada tweet relativo a la credibilidad del tweet y que se le pasa como parámetro al helper para transformarlo en %.

Mis Tweets
Este template es igual que el anterior la única diferencia reside en los tweets que se recuperan, que sólo son los del usuario.

```
let idUserT = parseInt(Meteor.user().services.twitter.id);
return Tweets.find({'user.id': idUserT}, {sort: {id: -1}});
```

Figura 35: Ejemplo de consulta

Al *find()* de la colección Tweets le pasamos el id de Twitter del usuario para buscar en la base de datos sólo los tweets cuyo *user.id* sea el del usuario. El resultado lo ordenamos por id de manera ascendente.

Sobre la app
Este template simplemente tiene información sobre la app, no tiene parte JavaScript.

4.2 Desarrollo de la aplicación, servidor

En la parte del servidor en la aplicación se establecen los métodos que se llaman desde la parte del cliente. También se establecen las *key* de acceso a la API de Twitter para que el cliente no tenga acceso a ellas.

En esta parte del servidor se ha definido el archivo *publications.js* que define las publicaciones que se quieren enviar al cliente con los trozos de la bases de datos que tiene que tener acceso.

```
Meteor.publish("tweets",function(){
    if (!this.userId) {
        return this.ready();
    }else{
        var idUserT = parseInt(Meteor.user().services.twitter.id);
        return Tweets.find({'user.id': idUserT}, {sort: {id: -1}});
    }
}),
```

Figura 36: Ejemplo de publicación

La manera de crear publicaciones es como se muestra en la imagen (Figura) donde se comprueba si hay usuario activo y si es así se devuelve la consulta necesaria, en este caso los tweets del propio usuario. Con esta publicación, el usuario en la parte cliente sólo tendrá acceso a lo que se devuelva desde aquí, mediante suscripciones, como se ha explicado anteriormente.

De la parte del cliente con código en la parte del servidor sólo se han creado para el registro / acceso y los tweets.

En este archivo se ha definido la configuración necesario para establecer conexión con la API de Twitter que necesita el paquete Service-Configuration.

También se han definido 2 métodos mediante *Meteor.methods({})*, estos métodos son los que se llaman desde la parte JavaScript del cliente.

Un método, *info*, establece dentro del paquete twit, los access token de cada cliente conectado, estos access tokens son necesarios para tener acceso a la cuenta de Twitter del cliente y poder recuperar sus tweets. Estos acces tokens están en la base de datos guardados, ya que a la hora de hacer el registro es parte de la información que el paquete Accounts-Twitter devuelve. Estos access tokens no salen del servidor en ningún momento, por lo tanto el cliente nunca tiene acceso a ellos.

El otro método, *actualizaTweets*, se encarga de actualizar tanto los tweets del propio usuario como los de la home de Twitter del usuario.

Esto se realiza al hacer un acceso al sistema o al hacer clic al botón de actualizar de la barra superior. El motivo de hacerlo de esta manera es que dependiendo de la cuenta del usuario, si la cantidad de tweets es muy alta, podría hacer que Twitter capase la conexión ya que tiene limites a la hora de devolver tweets. Otro motivo es que si estuviese actualizándose constantemente al usuario no le daría tiempo a ver los tweets que se vayan descargando.

Esta actualización siempre se realiza desde el último tweet descargado.

```
//Actualiza Tweets cada vez que se llama a la funcion (al iniciar sesión o hacer click en el boton de actualizar)
var ultimoID = Tweets.find({"user.id": parseInt(Meteor.user().services.twitter.id)}, {sort: {id: -1}, limit: 1}).fetch();
T.get('statuses/user_timeline', { screen_name: username, since_id: ultimoID[0].id, tweet_mode: 'extended'},  Meteor.bindEnvironment(function (err, data, response) {
    if(err){
        console.log("Error");
    }else{
        data.forEach(function(item, indes, array){
            item.reliability = 'pending';
            Tweets.update({id: item.id}, item, {upsert: true});
        })
    }
}))
```

Figura 37: Ejemplo de llamada a API de Twitter

Mediante,

var ultimoID = Tweets.find({"user.id": parseInt(Meteor.user().services.twitter.id)}, {sort: {id: -1}, limit: 1}).fetch();

se recupera el último id almacenado y con la llamada a la API de Twitter mediante T.get… se recuperan los tweets nuevos desde el id pasado como parámetro en *since_id*.

Con los tweets de la home de Twitter del usuario es igual, la diferencia es que el último id se recupera del documento *UsersTwitter*, donde se guardaba una lista de los tweets de la home de Twitter del usuario.

A la hora de almacenar los tweets en vez de utilizar un *insert*, se utiliza un *update* con un parámetro llamado *upsert*. Lo que permite este parámetro es que si el registro no existe, lo creo y si no se actualiza, siempre y cuando haya algún cambio a realizar si no, no hace ninguna modificación. Con esto se consigue que los tweets no se repitan dentro de la base de datos.

Por último también se define una función propia de Meteor llamada *Accounts.onCreateUser()*, esta función es llamada cuando un usuario se crea y es la

encargada de coger toda la información que devuelve Twitter para almacenarla en la base de datos, a parte se le han añadido la modificación de los tokens para poder tener acceso a los tweets de la cuenta de usuario y las llamadas a la API para poder descargar los tweets del usuario. Se ha establecido que al crear la cuenta del usuario se descarguen 100 tweets propios del usuario y 100 de la home de Twitter del usuario.

4.3 Desarrollo del servidor

Toda la aplicación necesita estar alojada en un servidor para que se puede utilizar desde cualquier parte en cualquier dispositivo.

Para el despliegue de esta aplicación se ha utilizado un servidor con Ubuntu 17.10.

En este servidor se han añadido los siguientes programas para que la aplicación funcione correctamente:

- Meteor
- NodeJS
- MongoDB
- Apache2

A parte del código de la aplicación.

Meteor tiene su propia instancia de MongoDB, pero se puede utilizar sin problemas una exterior.

El proceso de implementación en el servidor esta detallado paso por paso en el anexo.

5 Integración, pruebas y resultados

Las pruebas llevadas a cabo en esta aplicación se han realizado durante el desarrollo y una vez concluido el desarrollo.

La ventaja de utilizar Meteor es que al compartir código por parte de la aplicación y web, la comprobación es la misma para ambas partes.

Meteor al instalarse viene con 2 paquetes por defectos llamados *autopublish* e *insecure*.

Estos paquetes permiten tener acceso a la parte del servidor directamente desde la consola del navegador, lógicamente existen y vienen por defecto con la intención de realizar pruebas y ver los datos que va devolviendo la base de datos (Figura). Es prácticamente obligatorio que una vez se finalice el desarrollo desinstalar estos paquetes.

El paquete autopublish se encarga de publicar todas las colecciones del servidor en la caché del cliente, de este modo tienes toda la base de datos en el cliente y puede saber que datos necesitas recuperar a la hora de implementar las publicaciones y suscripciones.

El paquete insecure se encarga de permitir que desde el cliente se puedan utilizar todos los métodos que se pueden usar en MongoDB como inserts, updates, remove.

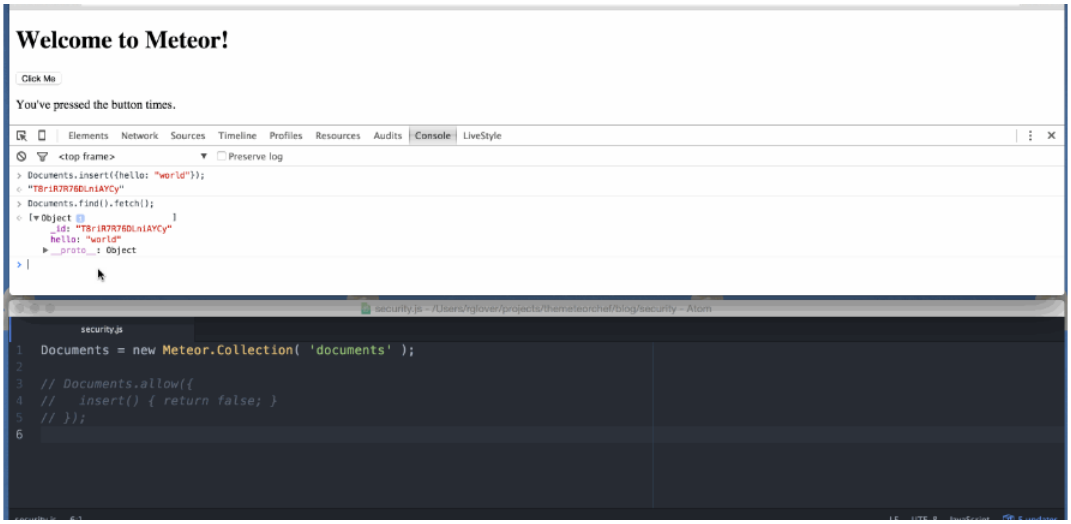

Figura 38: Muestra de lo que es posible hacer con los paquetes Autopublish y Insecure

Con estos paquetes se pueden realizar test durante se esta realizando el desarrollo de la aplicación.

También se han realizado pruebas de cada método de la parte del servidor de la aplicación desde varias cuentas distintas para comprobar que a cada usuario le aparece la información correcta y no existen cruces entre cuentas de usuario.

Y por último se han realizado pruebas una vez se ha hecho el despliegue de la aplicación en el servidor para comprobar que no había ningún problema de compatibilidad a la hora de realizar las instalaciones.

6 Conclusiones y trabajo futuro

6.1 Conclusiones

La realización de este proyecto ha sido una gran experiencia, no sólo por tener que aprender y realizar investigación sobre la plataforma utilizada, que no la había utilizado nunca, si no por comprobar cómo a medida que se iba desarrollando, iba cogiendo forma y se adaptaba todo de manera perfecta y sin complicaciones.

Gracias al desarrollo de este proyecto he podido aprender el uso de plataformas para realizar aplicaciones híbridas y comprobar el potencial que tienen frente a desarrollarlas de de manera nativa. He comprobado que Meteor facilita mucho el trabajo para desarrollar aplicaciones móviles. También me ha permitido utilizar MongoDB y ver la facilidad de poder implementar una base de datos, al igual que su utilización en Meteor.

He comprobado que la documentación oficial sirve de gran ayuda, sobretodo si es excelente y está bien explicada, como la de Meteor, ya que te facilita la comprensión cuando tienes que aprenderlo desde 0.

Me ha permitido descubrir otras maneras de desarrollar páginas web diferentes a como las solía hacer.

También he aprendido a trabajar con la API de Twitter, que tampoco la había utilizado nunca.

6.2 Trabajo futuro

Esta aplicación se ha realizado para un Trabajo de Final de Grado, aunque es funcional se puede mejorar, como todo.

Como ideas de mejoras que se podrían implementar en un futuro estarían:

- Poder poner un buscador por si en determinado momento necesitar buscar algún tweet de tu propia cuenta.
- Poder poner un buscador para buscar tweets mediante hastags.
- Poder realizar la comprobación de credibilidad de un tweet sin necesidad que sea uno de tu cuenta, es decir, a la hora de realizar una búsqueda por hastag.
- Poder ver la credibilidad de cuentas de usuario en vez de tweets, realizando alguna especie de media de la credibilidad que tienen los tweets de la cuenta.
- Poder comprobar de alguna manera el motiva de la puntuación de credibilidad que se le otorga a un tweet.

7 Bibliografía

[1] Twitter multiplica el impacto de las ciencias contra las ‘fake news’. http://www.agenciasinc.es/Noticias/Twitter-multiplica-el-impacto-de-las-ciencias-contra-las-fake-news

[2] Twitter endurece medidas contra las ‘fake news’. https://expansion.mx/tecnologia/2018/02/21/twitter-impedira-el-uso-de-bots-para-propagar-noticias-falsas

[3] La nueva herramienta de Facebook que va contra las ‘fake news’. http://www.elfinanciero.com.mx/tech/la-nueva-herramienta-de-facebook-que-va-contra-las-fake-news

[4] Facebook y Google estrenaron herramientas contra las ‘fake news’. https://www.clarin.com/sociedad/facebook-google-estrenaron-herramientas-fake-news_0_r1EExeN6x.html

[5] La herramienta de Facebook contra ‘fake news’. https://www.unotv.com/noticias/portal/negocios/detalle/la-herramienta-de-facebook-contra-fake-news-989926/

[6] Aplicación móvil. https://es.wikipedia.org/wiki/Aplicaci%C3%B3n_m%C3%B3vil

[7] No sin mi móvil: un breve repaso a su historia. https://www.nobbot.com/pantallas/movil-repaso-historia/

[8] Statistics and Market Data on Mobile Internet & Apps. https://www.statista.com/markets/424/topic/538/mobile-internet-apps/

[9] Tipos de aplicaciones móviles: nativas, webs, híbridas. https://www.solbyte.com/blog/2014/07/21/tipos-de-aplicaciones-moviles-nativas-webs-hibridas/

[10] Fake news thrives on social media. http://www.dailytexanonline.com/2016/11/21/fake-news-thrives-on-social-media

[11] Meteor. https://www.meteor.com

[12] En Twitter, las 'fake news' viajan más rápido y tienen más alcance que las noticias reales. https://www.nacion.com/tecnologia/redes-sociales/en-twitter-las-fake-news-viajan-mas-rapido-y/G7PBUOUHGBATJNT6OXG4WEJ54I/story/

8 Glosario

APP: Aplicación móvil

API: Application Programming Interface

HTML: Lenguaje de Marcado para Hipertextos

CSS: Cascading Stylesheets

JavaScript: Lenguaje de programación interpretado. Implementa la lógica del lado del cliente y del servidor.

JSON: JavaScript Object Notation

OAUTH: Open Authorization

DDP: Distributed Data Protocol

REST: Representational State Transfer

9 Anexos

9.1 Manual de instalación en servidor

A continuación se detallará como realizar la implementación del sistema paso a paso en un servidor para su utilización.

Esta explicación se basa en un servidor con Ubuntu 17.04. Para la instalación en otros sistemas operativos tanto las instalaciones como las rutas de los archivos pueden varias.

Antes que nada, para llevar a cabo esta implementación hay que realizar una serie de instalaciones previas:

- NodeJS
- MeteorJS
- MongoDB
- Apache

Puesto que el sistema utilizado es Ubuntu, la instalación y configuración de estas herramientas se realiza mediante consola.

Instalación NodeJS

Primero de todo se realiza la instalación de NodeJS que es el interprete de JavaScript y el núcleo de Meteor.

```
curl -sL https://deb.nodesource.com/setup_8.x | sudo -E bash -
sudo apt-get install -y nodejs
```

Instalación MongoDB

A continuación, se instala la base de datos MongoDB.
En este punto cabe destacar que Meteor implementa su propia instalación de MongoDB por lo cual este paso no es necesario realizarlo, de todos modos se explica por si se da el caso que se quiera tener la base de datos externa a Meteor. Más adelante esto hay que tenerlo en cuenta, ya que a la hora de configurar la aplicación hay que definir donde se encuentra MongoDB, si dentro del propio Meteor o externamente.

Lo primero de todo es importar la clave pública MongoDB utilizada por APT (Advanced Packaging Terminal) en cuyo caso le permite verificar el paquete.

```
sudo apt-key adv --keyserver hkp://keyserver.ubuntu.com:80 --recv
0C49F3730359A14518585931BC711F9BA15703C6
```

Seguidamente se crea el archivo de lista necesario para Ubuntu.

```
echo "deb [ arch=amd64,arm64 ] http://repo.mongodb.org/apt/ubuntu
xenial/mongodb-org/3.4 multiverse" | sudo tee
/etc/apt/sources.list.d/mongodb-org-3.4.list
```

Con esto ya se puede proceder a la instalación de MongoDB.

```
sudo apt update && sudo apt install mongodb-org -y
```

Para que MongoDB arranque con el sistema es necesario añadirlo como servicio.
Para ello se ejecuta el siguiente comando para generar un archivo systemd.

```
sudo nano /etc/systemd/system/mongodb.service
```

Dentro de este archivo se escribe el siguiente código.

```
[Unit]
Description=High-performance, schema-free document-oriented database
After=network.target
[Service]
User=mongodb
ExecStart=/usr/bin/mongod --quiet --config /etc/mongod.conf

[Install]
WantedBy=multi-user.target
```

Guardar el archivo y ejecutar la instancia de MongoDB.

```
sudo systemctl start mongodb
```

Se puede ver el estado del servicio para ver si esta ejecutando con el siguiente comando.

```
sudo systemctl status mongodb
```

Instalación Apache2

En caso que Apache2 no esté instalado es necesario hacerlo mediante el siguiente comando.

```
sudo apt update && sudo apt install apache2
```

Y permitir el acceso a puertos externos.

```
sudo ufw allow 'Apache Full'
```

Instalación MeteorJS

Ahora se procederá a la instalación de la propia plataforma que hace posible esta aplicación. La instalación en Ubuntu es realmente sencilla, basta con ejecutar un comando que se encarga de todo el proceso y lo deja todo lista para su funcionamiento.

```
curl https://install.meteor.com/ | sh
```

Sólo queda copiar la carpeta del proyecto en el servidor, en algún directorio que luego se recuerde, pues será necesario.

Con esto ya está todo listo e instalado para poder continuar con la implantación del proyecto en el servidor.
Lo que queda es la configuración del proxy inverso que proporciona Apache para el despliegue de la aplicación. Para eso se ejecuta el siguiente comando que instalará el módulo y lo pone disponible para Apache.

```
sudo apt-get install libapache2-mod-proxy-html libxml2-dev -y
```

Una vez instalado es necesario habilitar todos los módulos que Apache necesita para ejecutarse.

```
sudo a2enmod proxy
sudo a2enmod proxy_http
sudo a2enmod proxy_ajp
sudo a2enmod rewrite
sudo a2enmod deflate
sudo a2enmod headers
sudo a2enmod proxy_balancer
sudo a2enmod proxy_connect
sudo a2enmod proxy_html
```

Por defecto Apache crea un archivo de inicio predeterminado, este se tiene que desactivar, de lo contrario la aplicación predeterminada de Apache podría prevalecer sobre la que se está creando.

```
sudo a2dissite 000-default
```

Ahora se debe crear el archivo de host virtual.

```
sudo nano /etc/apache2/sites-available/<nombreDelProyecto>
```

Donde se debe sustituir <nombreDelProyecto> por el nombre del proyecto que se está configurando. Y dentro de este archivo hay que poner el siguiente texto.

```
<VirtualHost *:80>
  ErrorLog ${APACHE_LOG_DIR}/error.log
  CustomLog ${APACHE_LOG_DIR}/access.log combined
  ProxyPreserveHost On
  # Servers to proxy the connection, or
  # List of application servers Usage
  ProxyPass / http://0.0.0.0:3000/
 ServerName localhost
</VirtualHost>
```

Lo único a tener en cuenta en este archivo es el parámetro ProxyPass, donde se deberá cambiar http:/0.0.0.0:3000/ por la IP del servidor donde esté alojado, manteniendo el puerto

3000 que es el que por defecto utiliza Meteor. Una vez modificado se guarda el archivo.

Con esto ya está prácticamente todo configurado, en este punto, estando dentro del directorio del proyecto y escribiendo en la consola Meteor, debería arrancar la aplicación y debería ser accesible. Pero esta configuración mantiene Meteor arrancado mientras el servidor se mantenga encendido, en el momento en que se apague, Meteor y por consecuencia el proyecto dejarán de estar activos. Para solucionar esto hay que poner el proyecto en modo de producción, para eso se debe crear un servicio systemd, tal y como se hizo con MongoDB.

```
sudo nano /etc/systemd/system/<nombreDelProyecto>.service
```

Donde se debe sustituir <nombreDelProyecto> por el nombre del proyecto que se está configurando. Y dentro de este archivo hay que poner el siguiente texto.

```
[Service]
WorkingDirectory=/home/<nombreUsuario>/<nombreDelProyecto>
ExecStart=/usr/local/bin/meteor --production
Restart=always
StandardOutput=syslog
StandardError=syslog
SyslogIdentifier=<projectName>
User=<yourusername>
Environment=NODE_ENV=production
Environment=PWD=/home/<nombreUsuario>/<nombreDelProyecto>
Environment=PORT=3000
Environment=HTTP_FORWARDED_COUNT=1
```

En el parámetro WorkingDirectory hay que especificar el directorio donde se encuentra el proyecto.
En SyslogIdentifier hay que especificar el nombre del proyecto.
En User, el nombre del usuario que correrá la aplicación.
El parámetro PWD tiene que ser igual que WorkingDirectory.

En caso de que se desee utilizar MongoDB externo, es decir, el que no trae por defecto la instalación de Meteor, se deberá añadir a este archivo la siguiente linea.

```
Environment=MONGO_URL=mongodb://127.0.0.1:27017/Meteor
```

Con esta linea se especifica Meteor que coja como base de datos por defecto la instalación que se ha hecho al principio de esta guía y no la que trae el propio Meteor.

Por último se tiene que habilitar el servicio y arrancarlo.

```
sudo systemctl enable <nombreDelProyecto>.service
sudo systemctl start <nombreDelProyecto>.service
```

Con esto, queda implementado el proyecto en el servidor para que esté activo aunque el servidor se reinicie.

Como información se puede manejar el servicio para Meteor mediante los siguientes comandos.

Reiniciar la aplicación

```
sudo systemctl restart <nombreDelProyecto>
```

Parar la aplicación

```
sudo systemctl stop <nombreDelProyecto>
```

Ver el estado de la aplicación

```
sudo systemctl status <nombreDelProyecto>
```

9.2 Manual de creación .apk en Android

A continuación se detallará como crear el .apk para poder instalar la aplicación en un dispositivo móvil.

Para empezar hay que destacar que para poder crear el .apk tiene que estar instalado el sdk de android en Meteor mediante *meteor install-sdk android* y la plataforma de Android añadida mediante *meteor add-platform android*.

Una vez todo listo para que la aplicación pueda funcionar en dispositivos móviles tiene que estar firmada, si no el dispositivo no deja realizar la instalación.

La firma se realiza mediante el siguiente comando:

keytool -genkey -v -keystore <nombreApp>.keystore -alias <nombeApp> -keyalg RSA -keysize 2048 -validity 10000

Con esto creamos las claves y las almacenamos en el almacén de claves.

Ahora construimos el paquete de la aplicación mediante:

meteor build ../new_package --server http://xxx

../new_package – es la carpeta donde se va a construir el paquete de la aplicación
--server http://xxx: es el servidor con el que va a funcionar la aplicación, en caso de este proyecto http://homero.ii.uam.es:3000

Una vez creado el paquete hay que navegar hasta la carpeta creada (new_package)/android/project/build/outputs/apk/release

Dentro de esta carpeta hay que copiar el .keystore generado en la primera instrucción.

Y ejecutar:

jarsigner -verbose -sigalg SHA1withRSA -digestalg SHA1 -keystore <nombreApp>.keystore android-release-unsigned.apk <nombreApp>

Y verificar la firma:

jarsigner -verify -verbose -certs android-release-unsigned.apk

Por último sólo queda generar el .apk firmado mediante:

zipalign -v 4 android-release-unsigned.apk <nombreApp>.apk

Esto dejará el .apk en el directorio donde nos encontramos.

Sólo falta instalarla en el dispositivo móvil.